Extended-Entity Relationship Model

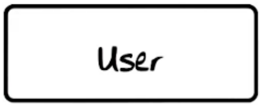
Notes for CS 6400: Database Systems Concepts & Design
Georgia Tech (Dr. Jay Summet with Dr. Leo Mark Christensen), Summer 2017
as recorded by Brent Wagenseller

Extended Entity Relationship Model Overview

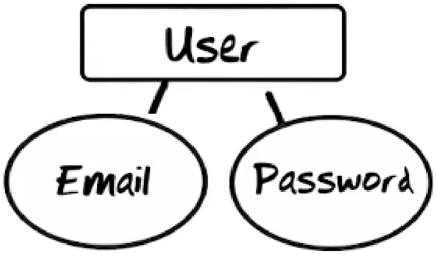
- The **Extended Entity Relationship Model** (**EER**) is a data model that is good at fixing and representing a perception of reality

EER Entity / Property Types

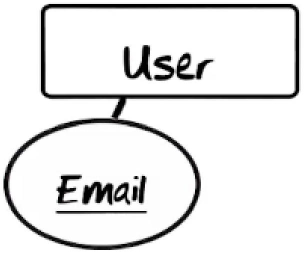
- Entity Type and Entity Surrogates
    - The **entity type** is a symbol in the EER which represents an entity
    - Example: User



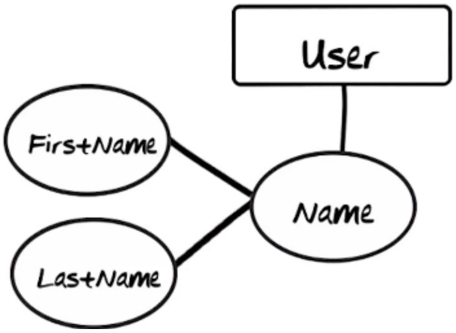
        - This ‘user’ entity type is a time-invariant representation of a set of users
    - One entity type represents multiple entities
    - Entity type names MUST be unique
- Single-Valued Properties



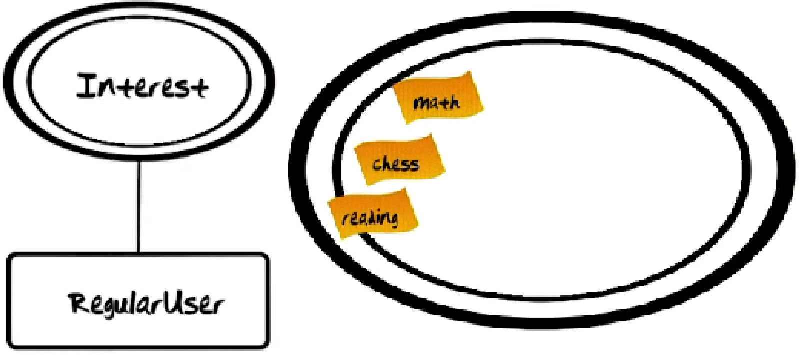
    - **Property types** are represented by ellipses
        - There are two properties in the example above
    - Single-line ellipses signifies it’s a single-value property
    - Property values can be
        - Lexical (text)
        - Visible (Pictures)
        - Audible (sound recordings)
    - Property values are things that name other things
- Identifying properties



  - **Identifying properties** are properties that identify entities / rows
  - Identifying properties are underlined
  - Identifying properties are unique – there CANNOT be more than one entry with a particular value
  - EVERY entity MUST be uniquely referenceable
    - Examples: every user, every book, every movie, etc
- Composite Properties

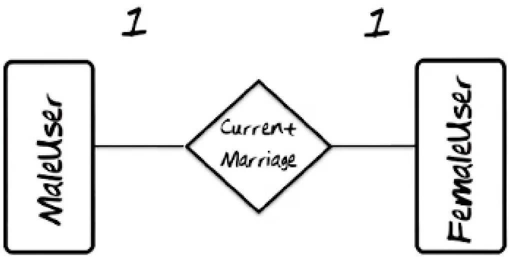


  - **Composite properties** are properties that are composed of smaller, sub-properties
  - EXAMPLE: a 'Name' property can be broken down into 'Last Name' and 'First Name'
- Multi-Valued Properties



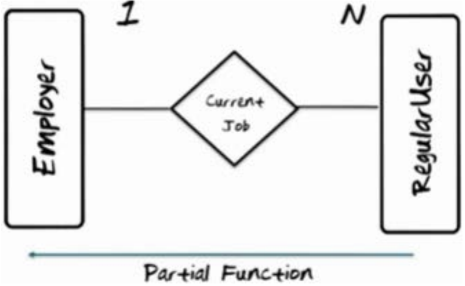
  - **Multi-valued properties** are properties that can have more than one entry for an entity type; they are represented by double ellipses
  - EXAMPLE: Users can have multiple interests

Extended Entity Relationship Model Relationship Types

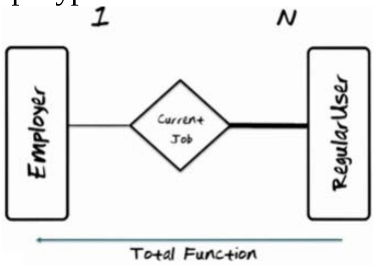


- The Basics
  - Relationship types are represented by diamonds.

    - The number above the connecting line is the cardinality (More on that below)
- 1-1 relationship Types
    - A **1-1 relationship type** is a relationship where one and only one entity type is connected to one – and only one – other entity type
        - In other words: the names of multiple relationship types between the same two entity types must be unique
    - EXAMPLE: marriage
    - This is also known as a **function** because it maps one element type to one – and only one – value in another entity type
    - A **partial function** is when there is a general 1-1 relationship between two entity types, but there are entities on either side that are unmatched
- 1-Many Relationship Types

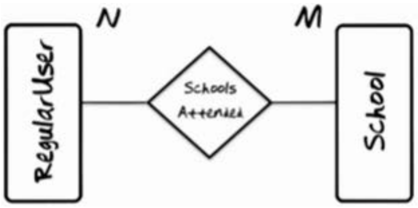


    - The **1-Many Relationship** is a relationship where one and only one entity type is connected to many entity types
    - 1-Many Relationship types can also be partial functions
        - Note this is true for ONLY the mapping of regular users to employers – NOT the other way around. This is because the definition of a function dictates that the variable entity (in this case users) can only be mapped to one and only one foreign entity
- Mandatory 1-N Relationship Types

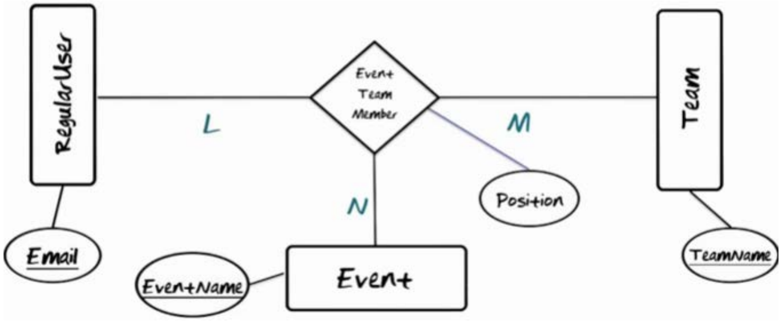


    - The **Mandatory 1-N Relationship** is a variation of the 1-Many Relationship
    - The bold line above signifies the user is REQUIRED to participate in a job
        - Note the line from employer to ‘current job’ is NOT bold, which signifies its NOT required
    - This is not only a partial function, but a **total function** as well, as ALL users are required to have a ‘current job’

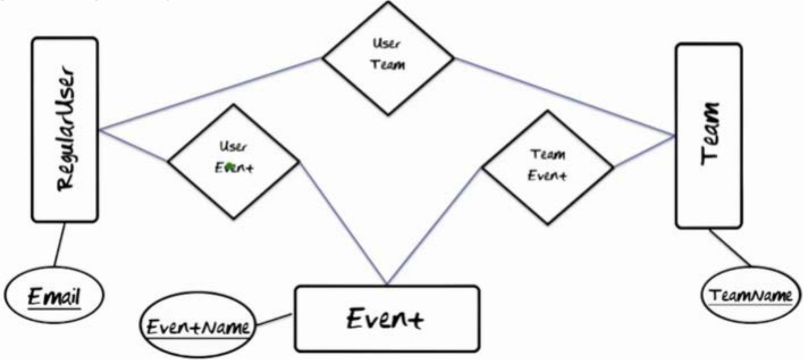
- N-M Relationships



  - The **N-M Relationship / Many-To-Many Relationship** is a relationship where many entity types are connected to many entity types
  - Note this is no longer a function; it’s a relationship (in the mathematical sense)
- N-ary Relationship Types

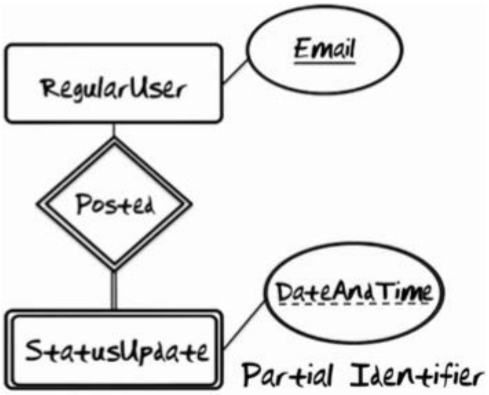


  - **N-ary relationship types** are relationships that tie in multiple entity types
  - Using the example above, an **instance** of the relationship ‘event team member’ <u>requires</u> a single regularUser (and their email), an Event (and an EventName), AND a Team (and a teamName)
    - Keep in mind that many users can belong to many events / teams and those events / teams can be involved in multiple teams / events, hence the many-many-many relationship
  - Typically, you will not see a relationship types higher than 2
    - They are difficult to understand and explain
    - Its not always possible to take an n-ary relationship type with n=3 or greater and decompose it into a conjunction of binary relationships
  - Re-organizing n-ary relationships

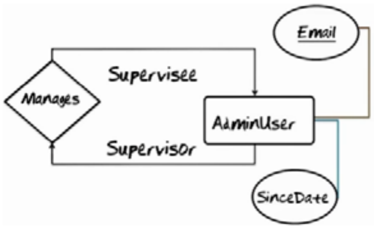


    - This revises a single n-ary relationship to multiple binary relationship types (using the example from before)

        - In this case, this breaks it down into 3 collections of 3 many-many binary (just two endpoint) relationships
            - It makes it easier to understand, but it doesn’t cover that a user can be on a particular team at a particular event – this is far more narrow than the multiple binary relationships above
- Identifying Relationships / Weak Entity Types
    - A weak **entity type** is when an entity type cannot stand on its own and relies on the identifier of another entity type
    - Example:



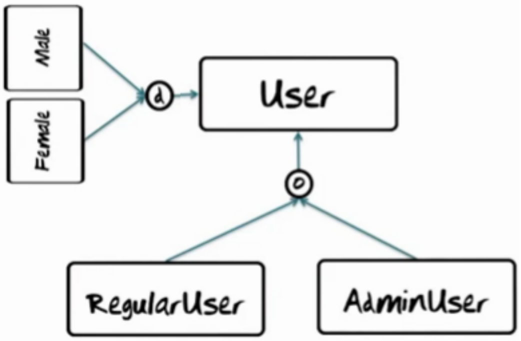
        - Users can post multiple status updates
        - Each status update has a datetime, but multiple users could theoretically post a status update at the SAME time, which would break the entity if datetime was the only identifier
            - Therefore, we ALSO use the ‘email’ as an additional identifier; its not possible for the same user to post multiple things at once (well, given one user can only be logged in from one device and we are measuring on milliseconds)
        - The above example is an **identifying relationship** type because in order to get a unique status update, we need to go through the email of RegularUser and combine with dateAndTime.
    - Note that the identifying relationship is signified with a double line around the relationship and the dependent entity type
    - Note that a **partial identifier** is a property that is needed to uniquely identify a relationship but cannot do so on its own; its signified with a dotted underline
        - In the example above, the partial identifier is ‘DateAndTime’
- Recursive Relationship Types
- A **recursive relationship type** relates an entity type to itself.
- Example:



- Notice that ‘roles’ are added to the relationship sides and direction on the lines are used
  - This is important in the example as we need to know who is the supervisor and who is the supervisee

Supertypes and Subtypes (aka is-a Relationship Types)

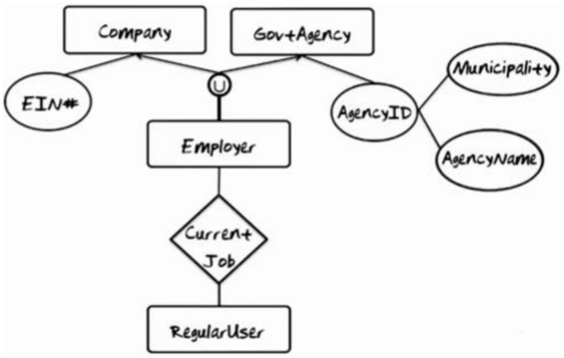
- A **subtype** is a subset of an entity type (in this case, the entity type is the **supertype**)
- The top level is typically more general than the lower tiers
- Its not enough to simply have a subtype, typically; the subtype generally needs to have unique, different properties that do not exist in other subtypes.
- For Example:



- Note the use of the small ‘d’; this is a disjointness constraint, and it signifies that the subtypes MUST be one or the other; they cannot overlap.
  - This can be handled by a **discriminator**, i.e. a field that lists a male OR female
  - BRENTS NOTE: Some schools of thought claim that you should ALWAYS have a discriminator field / column when the choices are disjoint
- Note the use of the large ‘D’; this allows the supertype to be a member of multiple subtypes simultaneously.
- Note the use of arrows
- Supertypes and Subtypes – Inheritance
  - Subtypes will ALWAYS inherit the properties of supertypes, but supertypes do NOT inherit properties of subtypes
- Union Entity Types
  - A **union entity type** encapsulates the intersection of multiple other types
    - It MUST be a ‘one or the other’ situation; one and only one type of the multiple types available to the union entity can be used
  - The union entity is signified by an encircled U
  - Rules for a union entity type
    - The upper tier entity MUST be comprised of a subset of the lower tiers
      - In the example below, Employer MUST either be a company OR a government agency
      - The subsets CANNOT overlap
        - In the example below, an employer cannot be a company AND a

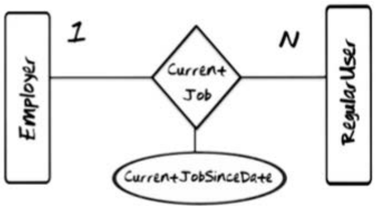
government agency

- ◦ Example:

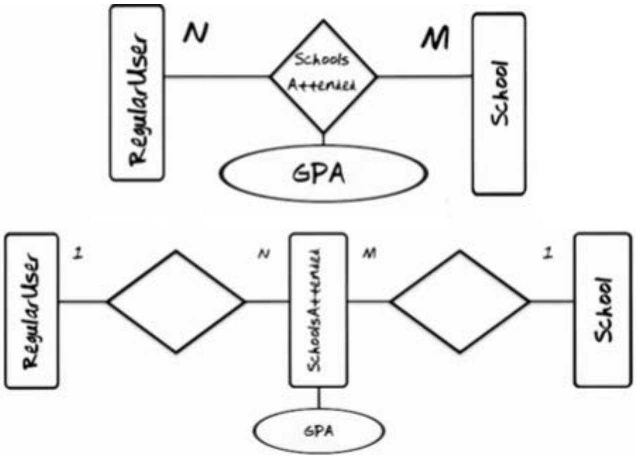


Are Relationships Entities

- Question: are relationships entities, or are they just 'glue' that ties together entities?
- The setup:



- Rules
  - Relationships may have attributes
  - For 1-N (and 1-1) relationships, attributes may be moved to the entity on the 'many-side' (either side)
- An objectified relationship type can also be deployed
  - An **objectified relationship type** is a relationship that is converted into an entity type; it acts as both a relationship and an entity!
  - If this is done, you need to model the two relationship types that are necessary in order to get the same functionality / cardinality as before
  - This is done by introducing two relationships
  - EXAMPLE



    - The top model excerpt can be converted to the bottom by objectifying the SchoolsAttended relationship.
    - The new objectified relationship results in a 1-M and M-1 relationship; overall, the objectified relationship retains the many-many relationship we had before
- BRENTS NOTE: I don't think Dr. Christensen directly answered, but it seems they are just 'glue'; the argument hinged on the relationship having a property, but he simply moved the property to the 'N' entity

Relationship Type of Entity Type?

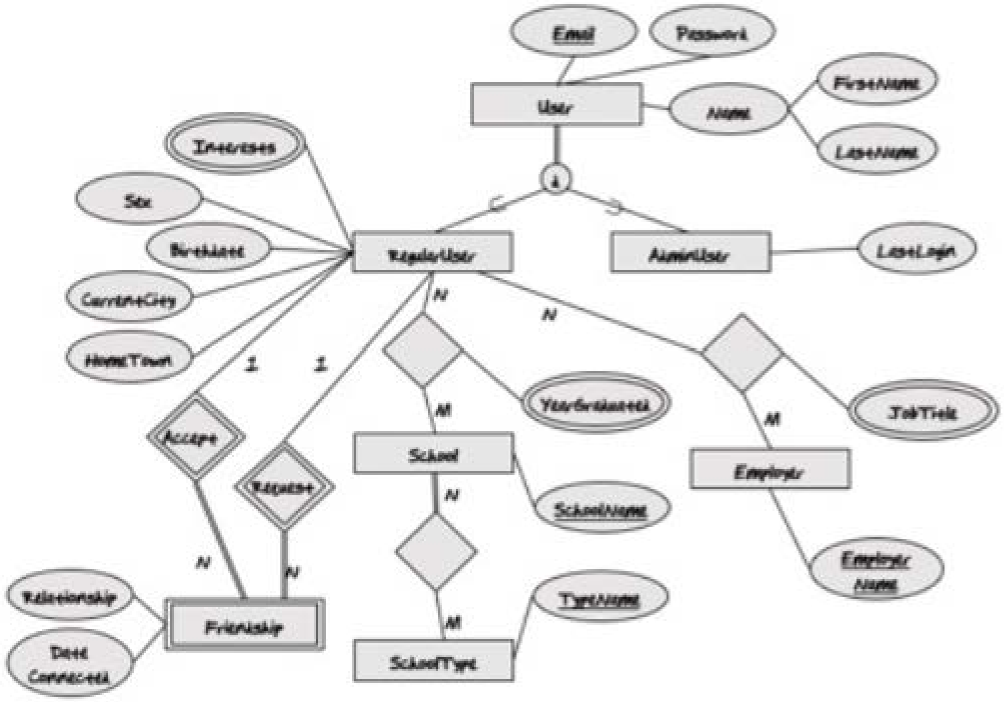
- Context is VERY important when you fix a perception of reality and want to represent it; nothing in reality stands out automatically and tells you if it's a relationship type, a property type, or an entity type.
  - In other words, use context when determining what is what for the 3 types

What can the EER Do?

- Three types of abstraction that are important when trying to fix a perception of reality:
  - Classification
    - EXAMPLE: The ability to define entity types
  - Generalization
    - EXAMPLE: Super/subtype relationship
  - Aggregation
    - Unfortunately – the entity relationship does NOT explicitly support aggregation
      - An example is a drivetrain of a car – there are different instances of component parts (engine, axle, shock, transmission); the drivetrain itself is ALSO an instance. This is difficult to aggregate
        - BRENTS NOTE: I am not quite sure I follow here, but I think what is being said is all of the components can be instanced (so they could represent different make/models of an engine, transmission, etc) which would mean you could have a (sort of) infinite number of combinations to make up a drivetrain. Since its very difficult to record EVERY possible permutation in a table without going to infinity, it can be said its difficult to 'aggregate' in an entity relationship (at least this is what I got from thelecture)
      - It can be 'forced' by using existing types

BRENTS NOTE: Full Example of an EER

- In the quizzes we were presented with a complete EER diagram; I wanted to show it here as I found it helpful when understanding the EER; so, without further ado:



What is the Result Type of a Query in an EER Diagram?

- <u>Queries do not actually have a 'type'; as such, there is no way that we can take the result of a query and continue to operate on it with a query language</u>
  - <u>This is because a query result is simply a list of properties, but this does not lend well to a rigid 'type' as found in the extended entity relationship model</u>
- Query languages have to be closed; it's the only way we can formulate high level ideas and ask high level questions
- <u>The lack of an agreed upon closed query language for the extended entity relationship model is the reason that database management systems are not based on the EER model</u>
  - There is no commercial database that implements EER directly
  - Almost all the commercial systems implement a relational data model
    - Therefore, we need to know how to map the EER to relations

Relational Model

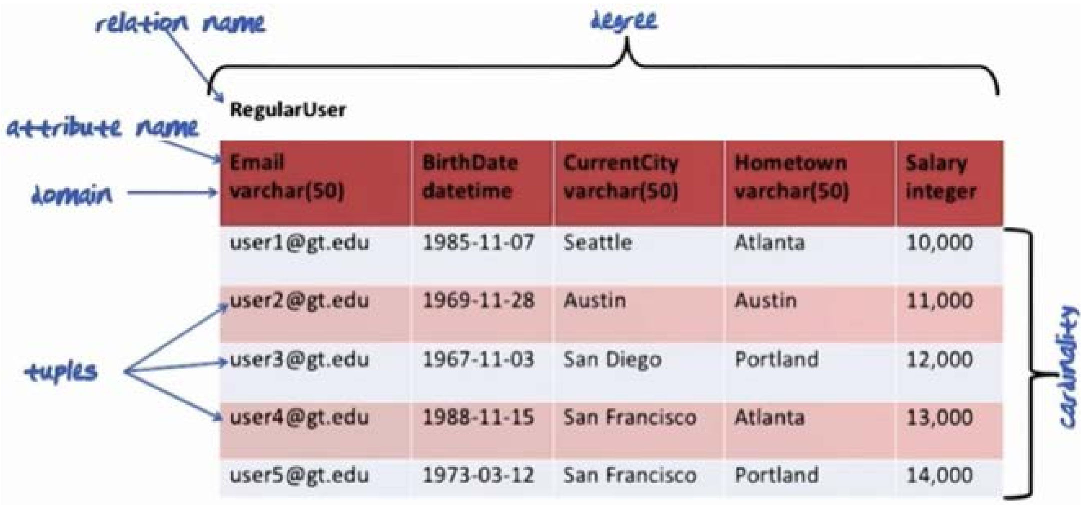
- When we look at a data model there are three things we must keep in mind:
  - Data structures
  - Constraints
  - Operations
    - Two fundamental notations that can be used to express operations in relational databases:

      - Relational algebra
      - Relational calculus
        - Two notations here:
          - Tuple Calculus
            - In **tuple calculus**, its tuples of relations that are variables
            - SQL is an example of a tuple calculus language
          - Domain Calculus
            - In **domain calculus**, its cells of domains that are variables
- Data Structures
  - As opposed to the EER and its multiple structures, **relational databases** only have one data structure: relations
    - **Relations** are a ‘catch all’, capturing property types, relationship types, and super sub types
  - A **domain**, D, is a set of atomic values
    - Atomic values have no meaning inside of it, according to a DBMS
      - Nothing is coded in the values; they are just values without meaning that we want to represent
      - We can think about a set of atomic values as a type
    - A domain appears to be data types (VARCHAR, CHAR, DATETIME, INT)
  - A **relation**, R, is a subset of the set of ordered n-tuples defined as

$$R \subseteq \{<d_1, d_2, \ldots, d_n> \mid d_i \in D_i, i=1, \ldots, n\}$$

    - Each one of the elements in the tuple (element D_i) is pulled from the corresponding domain D_i
      - Its critical to know that a relation is a set
    - A relation is a table
  - An **attribute**, A, is a unique name given to a domain in a relation
    - Attributes are used to explain or interpret the role of a domain in a relation
    - An attribute is a column in a table
    - An advantage of having unique names of attributes within a relation is now, instead of referring to the position if a column in a table, we can instead refer to it by name

- Tables
  - We illustrate relations by tables. Example:


relation name
degree
attribute name
domain
tuples
cardinality

RegularUser

Email varchar(50)	BirthDate datetime	CurrentCity varchar(50)	Hometown varchar(50)	Salary integer
user1@gt.edu	1985-11-07	Seattle	Atlanta	10,000
user2@gt.edu	1969-11-28	Austin	Austin	11,000
user3@gt.edu	1967-11-03	San Diego	Portland	12,000
user4@gt.edu	1988-11-15	San Francisco	Atlanta	13,000
user5@gt.edu	1973-03-12	San Francisco	Portland	14,000

        - Here, the relation name is ‘RegularUser’
        - There are 5 attributes (columns)
        - The domains are the data types
        - The number of attributes (columns) is the **degree**
        - The tuples are the rows (the ENTIRE row)
        - The **cardinality** is the number of total rows
    - <u>The value of a relation is independent of attribute order and tuple order – this is groundbreaking!</u>
        - <u>In other words: Because of attribute naming, the column order does not matter, and the row / record order does not matter when referencing the relation / table!</u>
- Constraints
    - Two types
        - Keys
        - Primary keys
    - **Primary keys** uniquely identify a tuple / row and are a bit more specific than keys
        - These enforce entity integrity and referential integrity
        - **Entity integrity** means the values of the primary key are explicitly unique, keep the same format (data type), AND cannot be NULL
        - **Referential Integrity** is, when a foreign table (read: a different table) uses the primary key of another table, the foreign key column MUST be a subset of the initial table’s primary key; the referencing table cannot use values in that field that do not exist in the original table’s primary key
    - BRENTS NOTE: Dr. Christensen does not expand on keys, but keys are the superset of a primary key; in other words, a ‘key’ can be a reference to a primary key, a secondary key, or a foreign key (I am sure we will discuss these other keys in a later lesson).